AFRL-IF-RS-TR-2002-273
Final Technical Report
October 2002

EVOLUTION OF SOFTWARE VIA ADAPTIVE PROGRAMMING

Northeastern University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. D886

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-273 has been reviewed and is approved for publication.

APPROVED: John J Crowter

JOHN J. CROWTER
Project Engineer

FOR THE DIRECTOR: Michael Talbert

MICHAEL L. TALBERT, Maj., USAF
Technical Advisor, Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 2002	Final Jun 96 – Nov 99

4. TITLE AND SUBTITLE
EVOLUTION OF SOFTWARE VIA ADAPTIVE PROGRAMMING

5. FUNDING NUMBERS
C - F30602-96-2-0239
PE - 62301E
PR - D886
TA - 01
WU - 01

6. AUTHOR(S)
Karl Lieberherr

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Northeastern University
423 Lake Hall
360 Huntington Avenue
Boston Massachusetts 02115

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2002-273

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: John J. Crowter/IFTB/(315) 330-2074/ John.Crowter@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	

13. ABSTRACT *(Maximum 200 Words)*
Adaptive Programming (AP) is a technology that improves the separation of traversal-related concerns by separating the concerns of where-to-go, what-to-do and when-to-do. The three concerns can be understood using the terminology of Aspect-Oriented Programming (AOP): pointcuts, advice and introductions. The concern what-to-do can be implemented as an advice and concern when-to-do as a pointcut. The concern where-to-go specifies a set of introductions that implement a set of traversal methods.

AP supports easy re-modularization of generic behavior that is formulated in terms of a generic class graph and traversal specifications (where-to-go concern) with respect to the class graph. Binding the generic behavior to a specific class graph involves updating the class graph and changing the traversal specifications, if needed. This high-level approach to re-modularization makes AP a useful tool to support software evolution.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Aspect-Oriented, Programming Adaptive Programming, Programming Languages, Software Development Environments, Software Design, Software Maintenance	26
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1.0 Introduction

Adaptive Programming (AP) is a programming technology where programs are split into crosscutting building blocks in a novel way to control tangling and redundancy. In AP, at least one of the building blocks is represented by a graph and other building blocks refer to subgraphs of the graph without revealing the details of the subgraphs. This is a form of succinct specification of graphs similar to the hierarchical specification of graphs widely used in hardware and software designs. A succinct specification can be expanded to a flat representation that is usually much larger than the succinct representation.

DARPA and Rome Laboratory have supported the work of the Demeter Research group since 1996. This report, written in April 2000 and updated in July 2002, summarizes the direction and accomplishments of our research over this period. Our results have been put on the web on a regular basis. In this final report, we summarize the most important information and provide links to the sources. The supported work falls in the domain of our research mission.

Mission of the Demeter Research Group

Our mission is to improve on current leading-edge software development practices using ideas from programming languages and software engineering. We build on technology that has industry acceptance, we propose improvements and we build tools and write papers to make the benefits of our improvements readily available. We strive for our technology to be self-evidently important and useful.

We teach the technology that we develop, after it has sufficiently matured, in graduate and advanced undergraduate courses and in tutorials at conferences. We distribute the software through the web to get feedback from external users.

Our focus (since 1985) is on Separation of Concerns (SOC) technology.

1. The job of a SOC technology is to turn a tangled and scattered implementation of a concern into a well-modularized implementation of the concern.

2. The job of an AOOSD (Adaptive Object-Oriented Software Development) technology, a special case of SOC, is to turn a tangled and scattered implementation of a concern into a well-modularized implementation of the concern that frees the programmer from details of the classes or methods where the implementation is scattered. The well-modularized concern is applicable to a family of diverse classes and uses some form of succinct representation, e.g., regular expressions or traversal specifications.

3. The job of an AOSD (Aspect-Oriented Software Development), a special case of SOC, is to turn a tangled and scattered implementation of a crosscutting concern into a well-modularized implementation of the concern.

The exact relationship between SOC, AOOSD and AOSD is still being worked out (see AOP in Demeter).

The mission has been successful and has led to major adoptions of our techniques in standard tools. Some examples: Xpath, JAXB, AspectJ, UML class diagrams. Our techniques have also been adopted in company specific tools, e.g., in a mission-critical tool at Verizon.

Research Group Members

The research group over this period has consisted of faculty member Karl Lieberherr (PI), Mira Mezini (University of Siegen and NU), Jens Palsberg (MIT and Purdue University), Boaz Patt-Shamir, Mitchell Wand. In support of the faculty have been students: Crista Lopes (supported by Xerox PARC), Doug Orleans, Kedar Patankar, Binoy Samuel, Linda Seiter, Johan Ovlinger, Joshua Marshall, and Geoff Hulten.

2.0 Research

Our most promising work is in the areas of collaborations (in the UML sense) and adapters. An OOPSLA 1998 paper (AP&PC Adaptive Plug&Play Components, http://www.ccs.neu.edu/research/demeter/biblio/components.html) describes the key ideas. Follow on papers are by Stefan Hermann and Mira Mezini at OOPSLA 2000 and by Mira Mezini and Klaus Ostermann at OOPSLA 2002. *Collaborations* express systems by starting with a basic system and adding more and more reusable enhancements (both functional and systemic) using *adapters*. Our work on adapters is described in the paper PCA. PCA stands for Pluggable Composite Adapter. AP&PC is integration work on Aspect-Oriented Programming and multi-dimensional separation of concerns. In AOP in Demeter we describe how we use AOP in Demeter in 5 different ways.

Adapters are made robust with respect to structural changes by using traversal strategies. Traversal strategies are a cornerstone of Adaptive Programming (AP). A patent that was applied for with a previous NSF grant has been refined with support by this grant (US Patent 5,946,490, issues Aug. 31, 1999). The work behind this patent provides an automata and graph theory of adaptive programming (Autonata and Graph Theory AP). More work on the theory of adaptive programming is referenced in the semantics section of the Demeter web page. A comparison of AP and object-oriented technology is also available.

To experiment with and evaluate the abstractions proposed, we built several tools. The most powerful one is DemeterJ but it also has a high learning curve. Therefore, we developed DJ, a less powerful and efficient but much easier to learn and use. The simplicity and ease of use helps to soften the learning curve associated with DemeterJ. Both DemeterJ and DJ use an implementation of traversal strategies that we have factored out into a separate AP Library .

The AP Library contains the core algorithm for expanding a succinct representation of a graph into a flat, detailed representation of a graph. A succinct specification is given by a strategy graph.

Definition: Given a graph G, a strategy graph S of G is any subgraph of the transitive closure of G. The flat representation of a strategy graph S with respect to G is (in many cases) the subgraph of G that consists of all paths in G that is expansions of paths in S.

The problem the AP Library solves is (in simplified form): Given as input a strategy graph S and a graph G, return the flat representation of S with respect to G. The flat representation is the set of paths in G defined by S and is called a traversal graph.

DemeterJ success at Verizon shows that DemeterJava is a powerful tool used by a Fortune 10 company in a mission-critical application. Demeter ideas have also been used in XML technology. We developed a pattern langauge for AP that facilitates the use of adaptive programming ideas.

We have also worked with Tendril Software Inc. who developed Structure Builder . Tendril Software was recently acquired by WebGain. StructureBuilder uses some of the ideas developed under this grant. See the joint paper: "Interaction Schemata: Compiling Interactions to Code" .

During the time of this grant, until the end of 1997, Cristina Lopes worked on her Northeastern PhD thesis on D and Aspect-Oriented Programming. She was supported by Xerox PARC with Gregor Kiczales as her co-advisor as well as the PI advisor. Her thesis work was the starting point of much of our most productive work on aspect-oriented programming.

3.0 Key Publications

The papers with publication dates from 1997 to 2000 have been supported by this DARPA grant and are available on the web at http://www.ccs.neu.edu/research/demeter/papers/publications.html. The abstracts of a selection of six important papers are listed below. A complete listing of papers can be found in Section 8, References.

3.1 "Adaptive Plug-and-Play Components for Evolutionary Software Development"

Mira Mezini and Karl Lieberherr

October 1998

Abstract

In several works on design methodologies, design patterns, and programming language design, the need for program entities that capture the patterns of collaboration between several classes has been recognized. The idea is that in general the unit of reuse is not a single class, but a slice of behavior affecting a set of collaborating classes. The absence of large-scale components for expressing these collaborations makes object-oriented programs more difficult to maintain and reuse, because functionality is spread over several methods and it becomes difficult to get the "big picture". In this paper, we propose Adaptive Plug and Play Components to serve this need. These components are designed such that they not only facilitate the construction of complex software by making the collaborations explicit, but they do so in a manner that supports the evolutionary nature of both structure and behavior.

Download from ftp://ftp.ccs.neu.edu/pub/people/lieber/appcs.pdf

3.2 "Programming with Aspectual Components"

KARL LIEBERHERR , DAVID LORENZ , and MIRA MEZINI

April 1, 1999

Abstract

Aspect-oriented programming (AOP) controls tangling of concerns by isolating aspects that crosscut each other into building blocks. Component-based programming (CBP) supports software development by isolating reusable building blocks that can be assembled and connected in many different ways. We show how AOP and CBP can be integrated by introducing a new component construct for programming class collaborations, called aspectual component. Aspectual components extend adaptive plug-and-play components (AP&P) with a modification interface that turns them into an effective tool for AOP. A key ingredient of aspectual components is that they are written in terms of a generic data model, called a participant graph, which is later mapped into a data model. We introduce a new property of this map, called instance-refinement, to ensure the proper deployment of components. We show how aspectual components can be implemented in Java, and demonstrate that aspectual components improve the AspectJ language for AOP from Xerox PARC.

Download from

http://www.ccs.neu.edu/research/demeter/papers/aspectual-comps/aspectual1.ps

3.3 "Traversals of Object Structures: Specification and Efficient Implementation"

Karl Lieberherr , Boaz Patt-Shamir

August 29, 1997

Abstract

Traversal of object structures is one of the ubiquitous routines in most types of information processing. In this paper we present a new approach, called strategies, to the task of traversing object structures. In our approach traversals are defined using a high-level directed graph description, which is compiled into a dynamic road map to assist run-time traversals. The complexity of the compilation algorithm is polynomial in the size of the strategy graph and the class graph of the given application. The implementation is practical and allows for dynamically creating and modifying the existing traversal strategy. A prototype of the system has been developed and is being successfully used. Previous approaches to traversal specifications were less general (corresponding to either a series-parallel or a tree graph), and their compilation algorithms were of exponential complexity in some cases. In an additional result we show that this bad behavior is inherent to the static traversal code generated by previous implementations, where traversals are carried out by invoking methods without parameters.

Download from

ftp://ftp.ccs.neu.edu/pub/people/lieber/strategies.ps

3.4 "The Refinement Relation of Graph-Based Generic Programs"

Karl Lieberherr and Boaz Patt-Shamir

September 1998

Abstract

This paper studies a particular variant of Generic Programming, called Adaptive Programming (AP). We explain the approach taken by Adaptive Programming to attain the goals set for Generic Programming. Within the formalism of AP, we explore the important problem of refinement: given two generic programs, does one express a subset of the programs expressed by the other? We show that two natural definitions of refinement coincide, but the corresponding decision problem is computationally intractable (co-NP-complete). We proceed to define a more restricted notion of refinement, which arises frequently in the practice of AP, and give an efficient algorithm for deciding it.

Download from
ftp://ftp.ccs.neu.edu/pub/people/lieber/graph-refine.ps

3.5. "Component Integration with Pluggable Composite Adapters"

Mira Mezini, Linda Seiter, and Karl Lieberherr

January 2000

Abstract

In this paper we address object-oriented component integration issues. We argue that traditional framework customization techniques are inappropriate for component-based programming since they lack support for non-invasive, encapsulated, dynamic customization. We propose a new language construct, called a pluggable composite adapter, for expressing component gluing. A pluggable composite adapter allows the separation of customization code from component implementation, resulting in better modularity, flexible extensibility, and improved maintenance and understandability. We also discuss alternative realizations of the construct.

Download from

http://www.ccs.neu.edu/home/lieber/s/compoint/composite-adapter.pdf

3.6. "Interaction Schemata: Compiling Interactions to Code"

Neeraj Sangal, Edward Farrell, Karl Lieberherr, David Lorenz

October 1998

Abstract

Programming object interactions is at the heart of object-oriented programming. To improve reusability of the interactions, it is important to program object interactions generically. We present two tools that facilitate programming of object interactions. StructureBuilder, a commercial tool, achieves genericity with respect to data structure implementations for collections, following ideas from generic programming, but focussing only on the four most important actions add, delete, iterate and find that are used to translate UML interaction diagrams into code. The focus of StructureBuilder is to generate efficient code from interaction schemata that are an improved form of interaction diagrams. DJ, a new research prototype intended for fast prototyping, achieves genericity with respect to the UML class diagram by dynamic creation of collections based on traversal specifications.

Download from

http://www.ccs.neu.edu/research/demeter/papers/generic-actions/tools99.ps

4.0 Teaching

The Demeter Seminar helped to disseminate the techniques as well as the courses Adaptive Object-Oriented Software and Advanced Object-Oriented Systems. The tools Demeter/Java and DJ are used in both courses. We developed tutorials related to Adaptive Programming both for ICSE 1997 and 2000.

Two PhD theses were completed with partial funding from this grant:

Linda Seiter

"DESIGN PATTERNS FOR MANAGING EVOLUTION", 1996,

and

Cristina Lopes

"A LANGUAGE FRAMEWORK FOR DISTRIBUTED PROGRAMMING, 1997.

5.0 Future Plans: The Tool Evolution Strategy of the Demeter Team

Following Alistair Cockburn (see his book on Agile Software Development), we view software development as a group game, which is goal seeking, finite and cooperative. We work together with our sponsors to produce working and useful systems.

Our mission is described at http://www.ccs.neu.edu/home/lieber/mission.html

The goal we seek is better separation of concerns. The game is finite from the point of each student (undergraduate or graduate student) who participates and from the point of view of each project which is funded for a finite time. But from the perspective of the Demeter Team, the game is "infinite" (for the next 15 years and hopefully much longer) in that each game that finishes should set the stage for the next game.

Currently we have three tools that we support:

DemeterJ	http://www.ccs.neu.edu/research/demeter/DemeterJava/
DJ	http://www.ccs.neu.edu/research/demeter/DJ/
DAJ	http://www.ccs.neu.edu/research/demeter/DAJ

DJ and DAJ are built on the AP Library which can be found at http://www.ccs.neu.edu/research/demeter/AP-Library/

Our current plan is to stop further development of DemeterJ and put the most useful features of DemeterJ into DAJ.

DemeterJ has served its purpose very well and introduced several innovations on top of the older Demeter/C++. but DemeterJ uses a non-standard programming language for

Adaptive Programming. In DAJ we found a way to express many of those capabilities just using a small extension to AspectJ. We basically introduce three new declarations in AspectJ: TraversalGraph, ClassGraph, Visitor and Behavior declarations.

DAJ will get the following capabilities:

a. Declare TraversalGraphs, ClassGraphs, Visitor and Behaviors.
Implemented using the AP Library.

b. Use class dictionaries and generate a parser and a print visitor.

c. Generate the other visitors that DemeterJ supports (Display Visitor, etc.).

d. Use XML schemas and generate code a la JAXB.

e. Use AspectJ as the weaver rather than the weaver currently in DemeterJ.

f. Any legal Java or AspectJ program will be a legal DAJ program.

DAJ is implemented using good separation of concerns at the cost of some minor user inconvenience. The DAJ implementation extends the AspectJ language with four kinds of declarations but it does so without depending on the internals of the AspectJ compiler. It only uses the published AspectJ language features. The AspectJ Team can modify their compiler anyway they want provided they maintain the interface.

The user inconveniences are:

i. DAJ users must put traversal-related concern declarations in separate files called *.trv.

ii. There is no traversal pointcut traversal(t) available for traversal t. Instead users must use call(* t(*)) and make sure t is not used outside the traversal.

While we gradually replace DemeterJ, we want to keep DJ around for the following reasons:

i. DJ provides for a quick introduction to programming traversal-related aspects just using ordinary Java.

ii. DJ allows for easy prototyping and testing of ideas, before they will be added to DAJ.

iii. DJ allows for dynamic adaptability not available in DAJ. DJ and DAJ can be used together in the same program.

Our plans for Eclipse (http://www.eclipse.org/):
The AspectJ Team is developing a plug-in for Eclispe.
We hope to build on that plug-in and add the DAJ specific capabilities:

Given a class graph in textual form (in black), we can highlight

1. the scope of a traversal (in red)

2. the scope of a behavior specification (traversal (in red) and visitor nodes (in blue) and edges (in green)) in this class graph.

If Eclipse has support for UML class diagrams, we can display the scope of a traversal graphically in the UML class diagram.

We are also planning to add Johan Ovlinger's tool on Aspectual Collaborations to Demeter's suite of tools.

6.0 Conclusion

This project made several contributions to the field of aspect-oriented software development:

(1) an efficient algorithm for implementing traversal related concerns (exponential improvement);

(2) a more succinct way to specify traversal-related aspects (exponential improvement);

(3) the concept of aspectual components that integrates the idea of aspects with the idea of components.

On the implementation side we developed software (DemeterJ) that is used by a Fortune 100 company in a mission critical application and our tools are used in numerous educational projects as well.

The technology we have developed is useful in many domains. The development of agent-based systems and its inherent complexities has come to the forefront of software systems in the past few years. The power and versatility of Aspect-Oriented Programming and Adaptive Programming paradigms are well suited for addressing the complexities of design and maintenance of such systems. We believe that the application of AOP and AP to agent-based systems will contribute greatly to design, implementation, and maintenance of these systems as well as software development in general.

7.0 References

A.V. Goldberg and K.J. Lieberherr, ***GEM: A generator of environments for metaprogramming***. SOFTFAIR II, ACM/IEEE Conference on Software Tools, San Francisco, CA, 1985, 86-95.

Karl Lieberherr and Ian Holland, ***Assuring Good Style for Object-Oriented Programs***. IEEE Software, September 1989, pages 38-48.

Karl Lieberherr, Irini Haralambopoulou, ***Adaptive and Conceptual Modeling of Software Development Environments***. Unpublished Manuscript, April 1992. 24 pages.

Karl Lieberherr, ***Component Enhancement: An Adaptive Reusability Mechanism for Groups of Collaborating Classes***. Information Processing '92, 12th World Computer Congress, Madrid, Spain, 179-185.

Karl Lieberherr, ***Experience with a Graph-Based Propagation Pattern Programming Tool***. International Workshop on CASE 1992, Montreal, Canada, 114-119.

Karl Lieberherr, Arthur Riel, ***Contributions to teaching object-oriented design and programming***. Proceedings of Conference on Object-Oriented Programming Systems, Languages and Applications, New Orleans, LA, October 1989, pp 11-22.

Karl Lieberherr and Walter Hürsch and Cun Xiao, ***Object-extending Class Transformations***. Formal Aspects of Computing, the International Journal of Formal Methods, 1994, pp 391-416.

Walter Hürsch and Linda M. Seiter, ***Automating the Evolution of Object-Oriented Systems***. International Symposium on Object Technologies for Advanced Software, Springer Verlag, Lecture Notes in Computer Science, Kanazawa, Japan, March 1996, pp 2-21.

Walter Hürsch and Cristina Videira Lopes, ***Separation of Concerns***. Northeastern University technical report NU-CCS-95-03, Boston, February 1995.

.

Ling Liu, Roberto Zicari, Walter Hursch, and Karl Lieberherr, ***Polymorphic Reuse Mechanisms for Object-Oriented Database Specifications***, IEEE Transactions on Knowledge and Data Engineering, 1997.

Cristina Videira Lopes and Karl Lieberherr, ***AP/S++: Case-study of a MOP for purposes of software evolution***. In proceedings of Reflection'96, S. Francisco, USA, April 1996.

Cristina Videira Lopes, "***Adaptive Parameter Passing"***. In proceedings of the 2nd International Symposium on Object Technologies for Advanced Software (ISOTAS'96), Kanazawa, Japan, March 1996. Springer-Verlag, Lecture Notes in Computer Science, n.1049.

Cristina Videira Lopes, ***Graph-based optimizations for parameter passing in remote invocations***. In proceeding of the 4th International Workshop on Object Orientation in Operating Systems (I-WOOOS'95), Lund, Sweden, August 1995. IEEE Computer Society Press.

Cristina Videira Lopes and Karl Lieberherr, ***Abstracting Process-to-Function Relations in Concurrent Object-Oriented Applications***. In proceedings of ECOOP'94, Bologna, Italy, July 1994. Springer-Verlag, Lecture Notes in Computer Science, n.821.

Cristina Videira Lopes and Karl Lieberherr, ***Generative Patterns***. ECOOP'94 Workshop on Patterns, Bologna, Italy, July 1994.

Jens Palsberg and Cun Xiao and Karl Lieberherr, ***Efficient Implementation of Adaptive Software***. ACM Transactions on Programming Languages and Systems, 1995.

Jens Palsberg and Boaz Patt-Shamir and Karl Lieberherr, ***A New Approach to Compiling Adaptive Programs***. Science of Computer Programming, 1997.

Linda M. Seiter and Jens Palsberg and Karl J. Lieberherr, ***Evolution of Object Behavior using Context Relations***. In proceedings of Symposium on Foundations of Software Engineering, SIGSOFT 1996 and IEEE Transactions on Software Engineering 1998.

Karl Lieberherr and Doug Orleans, ***Preventive Program Maintenance in Demeter/Java*** . In proceedings of International Conference on Software Engineering, ICSE 1997, Boston, MA, pages 604-605.

Karl Lieberherr and Boaz Patt-Shamir, ***Traversals of Object Structures: Specification and Efficient Implementation*** . Technical Report: NU-CCS-97-15, September 1997.

Mira Mezini and Karl Lieberherr, ***Adaptive Plug-and-Play Components for Evolutionary Software Development*** . Technical Report: NU-CCS-98-3, April 1998 and OOPSLA '98 paper.

Dean Allemang and Karl Lieberherr, ***Softening Dependencies between Interfaces*** . Technical Report: NU-CCS-98-07, August 1998.

Luis Blando and Karl Lieberherr and Mira Mezini, ***Modeling Behavior with Personalities*** . Technical Report: NU-CCS-98-08, August 1998 and SEKE 1999 paper.

Johan Ovlinger and Karl Lieberherr, ***Class Graph Views (Draft)*** . Technical Report: NU-CCS-98-09, August 1998.

Karl Lieberherr and Boaz Patt-Shamir, ***The Refinement Relation of Graph-Based Generic Programs*** . Lecture Notes in Computer Science, Springer Verlag, Proceedings of Dagstuhl Workshop on Generic Programming, David Musser at al.

Neeraj Sangal and Edward Farrell and Karl Lieberherr, ***Interaction Graphs: Object Interaction Specifications and their Compilation to Java*** . Technical Report, NU-CCS-98-11, Oct. 1998.

Luis Blando, ***Designing and Programming with Personalities*** . Master's Thesis, Technical Report: NU-CCS-98-12, December 1998.

Johan Ovlinger and Mitchell Wand, ***A Language for Specifying Traversals of Object Structures*** . Technical Report, NU-CCS-98, November 1998 and OOPSLA '99 paper.

Karl Lieberherr and David Lorenz and Mira Mezini, ***Programming with Aspectual Components*** . Technical Report, NU-CCS-99-01, March 1999.

Neeraj Sangal and Edward Farrell and Karl Lieberherr and David Lorenz, ***Interaction Schemata: Compiling Interactions to Code*** . TOOLS '99, Santa Barbara, CA, August 1999, IEEE Computer Society Press.

Linda Seiter and Mira Mezini and Karl Lieberherr, ***Dynamic Component Glue*** . First International Symposium on Generative and Component-Based Software Engineering, September 1999, Springer Verlag.

Mira Mezini and Linda Seiter and Karl Lieberherr, ***Component Integration with Pluggable Composite Adapters*** . Software Architectures and Component Technology: The State of the Art in Research and Practice, Mehmet Aksit, editor, Kluwer Academic Publishers, 2000.

Johan Ovlinger, ***Aspectual Collaborations and Modular Programming*** . NU-CCS-2000-04, 27 pages.

Karl Lieberherr and Doug Orleans and Johan Ovlinger, ***Aspect-Oriented Programming with Adaptive Methods*** . NU-CCS-2001-02, 15 pages.

Doug Orleans and Karl Lieberherr, ***DJ: Dynamic Adaptive Programming in Java*** . NU-CCS-2001-03, 10 pages. 8 page version in Reflection 2001.

Karl Lieberherr and Johan Ovlinger and Mira Mezini and David Lorenz, ***Modular Programming with Aspectual Collaborations*** . NU-CCS-2001-04, 12 pages.

Karl Lieberherr and Doug Orleans and Johan Ovlinger, ***Aspect-Oriented Programming with Adaptive Methods*** . CACM, October 2001, 3 pages.

Prasenjit Adak and Huichan He and Karl Lieberherr, ***Adaptive XML/Java Data-Binding*** . Summary of Master's Thesis by Huichan He in the College of Engineering, 10 pages.

Karl Lieberherr, ***Computer Science Adapts Techniques from Engineering and Mathematics*** .

Mitchell Wand and Karl Lieberherr, ***Traversal Semantics in Object Graphs*** . NU-CCS-2001-05, 15 pages.

Karl Lieberherr and David Lorenz and Doug Orleans and Johan Ovlinger and Mitchell Wand and Pengcheng Wu ***Demeter Aspects*** . OOPSLA 2001 Poster summarizing our research work.

Karl Lieberherr, ***Coupling Mechanisms in Aspect-Oriented Software*** . "NSF workshop: Software Design and Productivity (SDP), 5 pages.

Pengcheng Wu, ***Implementing Aspectual Collaborations with AspectJ*** . Technical Report NU-CCS-01-19.

Karl Lieberherr, David H. Lorenz, Johan Ovlinger ***Aspectual Collaborations for Collaboration-Oriented Concerns*** . Technical Report NU-CCS-01-08.

Doug Orleans, ***Incremental Programming with Extensible Decisions*** . First International Conference on Aspect-Oriented Software Development 2002, 9 pages.

Johan Ovlinger, Karl Lieberherr, David H. Lorenz, ***Aspects and Modules Combined*** . Technical Report NU-CCS-02-03.

8.0 Acknowledgements:

Thanks: I would like to thank DARPA and the Air Force Research Laboratory for the support of our work on Adaptive and Aspect-Oriented Programming. Many warm thanks to all participants in the project at Northeastern University and elsewhere who have made the project a success. I would like to thank John Crowter for his help with this final report.